# Contributing

<!-- gen:toc -->

- [How to Contribute](#how-to-contribute)

\* [Getting Code](#getting-code)

\* [Code reviews](#code-reviews)

\* [Code Style](#code-style)

\* [API guidelines](#api-guidelines)

\* [Commit Messages](#commit-messages)

\* [Writing Documentation](#writing-documentation)

\* [Adding New Dependencies](#adding-new-dependencies)

\* [Running & Writing Tests](#running--writing-tests)

\* [Public API Coverage](#public-api-coverage)

- [Contributor License Agreement](#contributor-license-agreement)

\* [Code of Conduct](#code-of-conduct)

<!-- gen:stop -->

## How to Contribute

### Getting Code

1. Clone this repository

```bash

git clone https://github.com/microsoft/playwright

cd playwright

```

2. Install dependencies

```bash

npm install

```

3. Build Playwright

```bash

npm run build

```

4. Run all Playwright tests locally. For more information about tests, read [Running & Writing Tests](#running--writing-tests).

```bash

npm test

```

### Code reviews

All submissions, including submissions by project members, require review. We

use GitHub pull requests for this purpose. Consult

[GitHub Help](https://help.github.com/articles/about-pull-requests/) for more

information on using pull requests.

### Code Style

- Coding style is fully defined in [.eslintrc](https://github.com/microsoft/playwright/blob/master/.eslintrc.js)

- Comments should be generally avoided. If the code would not be understood without comments, consider re-writing the code to make it self-explanatory.

To run code linter, use:

```bash

npm run eslint

```

### API guidelines

When authoring new API methods, consider the following:

- Expose as little information as needed. When in doubt, don’t expose new information.

- Methods are used in favor of getters/setters.

- The only exception is namespaces, e.g. `page.keyboard` and `page.coverage`

- All string literals must be small case. This includes event names and option values.

- Avoid adding "sugar" API (API that is trivially implementable in user-space) unless they're \*\*very\*\* common.

### Commit Messages

Commit messages should follow the Semantic Commit Messages format:

```

label(namespace): title

description

footer

```

1. \*label\* is one of the following:

- `fix` - playwright bug fixes.

- `feat` - playwright features.

- `docs` - changes to docs, e.g. `docs(api.md): ..` to change documentation.

- `test` - changes to playwright tests infrastructure.

- `devops` - build-related work, e.g. CI related patches and general changes to the browser build infrastructure

- `chore` - everything that doesn't fall under previous categories

2. \*namespace\* is put in parenthesis after label and is optional. Must be lowercase.

3. \*title\* is a brief summary of changes.

4. \*description\* is \*\*optional\*\*, new-line separated from title and is in present tense.

5. \*footer\* is \*\*optional\*\*, new-line separated from \*description\* and contains "fixes" / "references" attribution to github issues.

Example:

```

fix(firefox): make sure session cookies work

This patch fixes session cookies in firefox browser.

Fixes #123, fixes #234

```

### Writing Documentation

All public API should have a descriptive entry in [`docs/api.md`](https://github.com/microsoft/playwright/blob/master/docs/api.md). There's a [documentation linter](https://github.com/microsoft/playwright/tree/master/utils/doclint) which makes sure documentation is aligned with the codebase.

To run the documentation linter, use:

```bash

npm run doc

```

### Adding New Dependencies

For all dependencies (both installation and development):

- \*\*Do not add\*\* a dependency if the desired functionality is easily implementable.

- If adding a dependency, it should be well-maintained and trustworthy.

A barrier for introducing new installation dependencies is especially high:

- \*\*Do not add\*\* installation dependency unless it's critical to project success.

### Running & Writing Tests

- Every feature should be accompanied by a test.

- Every public api event/method should be accompanied by a test.

- Tests should be \*hermetic\*. Tests should not depend on external services.

- Tests should work on all three platforms: Mac, Linux and Win. This is especially important for screenshot tests.

Playwright tests are located in [`test/test.js`](https://github.com/microsoft/playwright/blob/master/test/test.js)

and are written with a [TestRunner](https://github.com/microsoft/playwright/tree/master/utils/testrunner) framework.

These are integration tests, making sure public API methods and events work as expected.

- To run all tests:

```bash

npm run test

```

- To run all tests in Chromium

```bash

npm run ctest # also `ftest` for firefox and `wtest` for WebKit

```

- To run tests in parallel, use `-j` flag:

```bash

npm run wtest -- -j 4

```

- To run tests in "verbose" mode or to stop testrunner on first failure:

```bash

npm run ftest -- --verbose

npm run ftest -- --break-on-failure

```

- To run a specific test, substitute the `it` with `fit` (mnemonic rule: '\*focus it\*'):

```js

...

// Using "fit" to run specific test

fit('should work', async ({server, page}) => {

const response = await page.goto(server.EMPTY\_PAGE);

expect(response.ok).toBe(true);

});

```

- To disable a specific test, substitute the `it` with `xit` (mnemonic rule: '\*cross it\*'):

```js

...

// Using "xit" to skip specific test

xit('should work', async ({server, page}) => {

const response = await page.goto(server.EMPTY\_PAGE);

expect(response.ok).toBe(true);

});

```

- To run tests in non-headless (headful) mode:

```bash

HEADLESS=false npm run ctest

```

- To run tests with custom browser executable, specify `CRPATH`, `WKPATH` or `FFPATH` env variable that points to browser executable:

```bash

CRPATH=<path-to-executable> npm run ctest

```

- To run tests in slow-mode:

```bash

HEADLESS=false SLOW\_MO=500 npm run wtest

```

- To debug a test, "focus" a test first and then run:

```bash

BROWSER=chromium node --inspect-brk test/test.js

```

- When should a test be marked with `skip` or `fail`?

- \*\*`skip(condition)`\*\*: This test \*should \*\*\*never\*\*\* work\* for `condition`

where `condition` is usually a certain browser like `FFOX` (for Firefox),

`WEBKIT` (for WebKit), and `CHROMIUM` (for Chromium).

For example, the [alt-click downloads test](https://github.com/microsoft/playwright/blob/471ccc72d3f0847caa36f629b394a028c7750d93/test/download.spec.js#L86) is marked

with `skip(FFOX)` since an alt-click in Firefox will not produce a download

even if a person was driving the browser.

- \*\*`fail(condition)`\*\*: This test \*should \*\*\*eventually\*\*\* work\* for `condition`

where `condition` is usually a certain browser like `FFOX` (for Firefox),

`WEBKIT` (for WebKit), and `CHROMIUM` (for Chromium).

For example, the [alt-click downloads test](https://github.com/microsoft/playwright/blob/471ccc72d3f0847caa36f629b394a028c7750d93/test/download.spec.js#L86) is marked

with `fail(CHROMIUM || WEBKIT)` since Playwright performing these actions

currently diverges from what a user would experience driving a Chromium or

WebKit.

### Public API Coverage

Every public API method or event should be called at least once in tests. To ensure this, there's a `coverage` command which tracks calls to public API and reports back if some methods/events were not called.

Run all tests for all browsers with coverage enabled:

```bash

npm run coverage

```

There are also per-browser commands:" `npm run ccoverage`, `npm run fcoverage` and `npm run wcoverage`.

## Contributor License Agreement

This project welcomes contributions and suggestions. Most contributions require you to agree to a

Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us

the rights to use your contribution. For details, visit https://cla.opensource.microsoft.com.

When you submit a pull request, a CLA bot will automatically determine whether you need to provide

a CLA and decorate the PR appropriately (e.g., status check, comment). Simply follow the instructions

provided by the bot. You will only need to do this once across all repos using our CLA.

### Code of Conduct

This project has adopted the [Microsoft Open Source Code of Conduct](https://opensource.microsoft.com/codeofconduct/).

For more information see the [Code of Conduct FAQ](https://opensource.microsoft.com/codeofconduct/faq/) or

contact [opencode@microsoft.com](mailto:opencode@microsoft.com) with any additional questions or comments.